Table of Contents

Simulation of the RPC Trigger Logic

Introduction

A set of classes has been written that allows users to easily plug into their analyses a simulation of the trigger logic and inefficiencies of the RPCs.

The Resistive Plate Chambers (RPCs) are the trigger chambers for the barrel of the muon spectrometer. The trigger is separated into projective towers defined by the cabling of the RPCs. In each tower, a trigger logic defining the number of gas gaps required in each of the RPCs is run. This suite of classes simulates that logic.

- `TriggerAlgorithm.cpp`

  is the basic building block class.
- `LowPtTrigger.cpp`

  is an example of a class that uses TriggerAlgorithm objects to represent a trigger logic. It can be used to simulate the low pt trigger of the RPCs.
- `InefficientTrigger.cpp`

  is a class that combines the running of the trigger logic with a parametric simulation of the RPC efficiency.

Any questions or concerns should go to Tomo Lazovich (tomo.lazovich@cernNOSPAMPLEASE.ch)

Code Structure

TriggerAlgorithm.cpp

This class is the basic building block of the trigger simulation structure. It allows you to define the number of gas gaps in a single trigger tower you require to be hit for each of the 3 RPCs.

How to initialize an object

Constructor takes 7 arguments. These arguments are, in order,

- # of gas gaps required for RPC1
- # of gas gaps required for RPC2
- # of gas gaps required for RPC3
- Require both eta and phi for RPC1?
- Require both eta and phi for RPC2?
- Require both eta and phi for RPC3?
- Width of road with which strips can be looked at (in strip #)

Example:

```
TriggerAlgorithm* trig = new TriggerAlgorithm(2, 1, 0, true, true, false, 1000);
```

How to invoke the trigger

The main function in this class is the following:

```
bool TriggerAlgorithm::trigger(vector<unsigned int> hit_ids)
```

This function takes a vector of unsigned integers corresponding to the hit_ids of RPC hits in the event, and returns true if there was a combination of hits in **any single trigger tower** that satisfies the trigger logic requirements. If there is no combination of hits that satisfies the trigger, it returns false.

Note: In general, you will probably not use the TriggerAlgorithm class directly. To represent a true trigger logic that we use, you create a wrapper class that combines 1 or more of these. An example of such a class is the LowPtTrigger.cpp, described below.

LowPtTrigger.cpp

This class creates two TriggerAlgorithm objects to represent the overall 3/4 logic that the RPC low pt trigger uses. A 3/4 logic can mean 2 gas gaps in RPC 1 and 1 in RPC 2, or 1 gas gap in RPC 1 and 2 in RPC 2. The LowPtTrigger class ORs the two TriggerAlgorithm objects defining these logics together.

How to initialize an object

The constructor takes 1 argument, the width of the road (in # of channels) used for the "pointing" part of the logic.

Example:

```
LowPtTrigger* low_pt = new LowPtTrigger(1000);
```

Note: The TriggerAlgorithm objects used to define this LowPtTrigger are statically defined in the constructor of LowPtTrigger. It is one example of any number of wrapper classes you can make to define different trigger logics.

How to invoke the trigger

The main function here is

```
bool LowPtTrigger::trigger(vector<unsigned int> hit_ids)
```

This function takes a vector of unsigned integers corresponding to the hit_ids of RPC hits in the event, and returns true if there was a combination of hits in **any single trigger tower** that satisfies the trigger logic requirements **for either of the TriggerAlgorithms defined in the constructor**. If there is no combination of hits that satisfies the trigger, it returns false.

Note: This class can be used to simulate the low pt trigger logic on its own. However, if you want to also simulate the inefficiencies, you must use InefficientTrigger.cpp

InefficientTrigger.cpp

This class combines the LowPtTrigger logic with a parametric simulation of the inherent RPC hardware inefficiencies.

How to initialize an object

The constructor here takes 3 arguments. They are as follows:

- the LowPtTrigger object used to define the logic
- a global inefficiency for the RPCs (in addition to the hardware efficiency)
- a seed for the random number generator used to simulate the efficiency

Example:

```
LowPtTrigger* low_pt = new LowPtTrigger(1000);
InefficientTrigger* trig = new InefficientTrigger(low_pt, 0.35, 17);
```

In the example above, this InefficientTrigger object will run a low_pt algorithm and lose 35% of the hits **in addition to** the hits lost due to hardware efficiency simulation.

How to invoke the trigger

There are many options for invoking the trigger both with and without the efficiency applied.

- `bool InefficientTrigger::shouldTrigger(vector<unsigned int> hit_ids)`

  This function applies the trigger logic defined by the LowPtTrigger object given in the constructor without losing any hits due to efficiency.
- `bool InefficientTrigger::doesTrigger(vector<unsigned int> hit_ids)`

  This function takes the vector of hit_ids and applies various efficiencies to it. First, it applies the hardware efficiency of the RPCs to the vector. Currently, the efficiencies are defined by hand on a run-by-run basis in the helper file rpcEfficiency.C. Very soon, though, the code will be changed to read in an efficiency map from a text file in order to define the efficiencies on a panel by panel basis. The function will lose hits from the vector with a probability corresponding to the inefficiency of the hardware. Then, it will lose hits with a probability corresponding to the global inefficiency defined in the constructor. Finally, it runs the trigger logic on the remaining hits and returns true if there is a combination among the hits left that satisfies the LowPtTrigger::trigger() function.

Additional useful functions

It is also possible to apply the inefficiencies mentioned without actually running the trigger logic. There are two functions that do this.

- `vector<unsigned int> InefficientTrigger::removeHits(vector<unsigned int> hit_ids)`

  This function removes hits only in chambers that were defined as completely off, and then returns the vector of the remaining hits.
- `vector<unsigned int> InefficientTrigger::removeHitsWithHardware(vector<unsigned int> hit_i`

  This function removes hits in all of the off chambers and also loses hits with a probability corresponding to the inefficiency of the chamber where the hit is located. It also applies the global inefficiency defined. It then returns the vector of remaining hits.

Usage Notes

shouldTrigger is what you would run on data if you were trying to use this class as an offline trigger confirmation. doesTrigger would be run on MC to confirm the trigger if you wanted the inefficiencies of the RPCs in data to be included in the trigger running on Monte Carlo.

Pending Changes

- Will soon allow user to define efficiency panel by panel in a text file and have this file parsed in order to define the efficiencies.
- The program currently uses a static cabling map to define the trigger towers. Ultimately, it would be ideal to have this program import the most recent cabling map at runtime.

I am very open to feature requests. If you have any questions or concerns, please email me at tomo.lazovich@cernNOSPAMPLEASE.ch

-- TomoLazovich - 30 Jul 2009

This topic: Sandbox > TomoLazovichSandbox
Topic revision: r3 - 2009-07-31 - TomoLazovich